Master's Thesis

Case Study in Software Testing or Software Analytics (focus on software quality)

Supervisor: Dietmar Pfahl (dietmar dot pfahl ät ut dot ee)

This is a "placeholder" Master's Thesis project topic, which needs to be negotiated individually. If you work in a IT company and you are actively engaged in a software testing or software analytics, or if you can convince your hierarchy to put in time and resources into such a project in the near-term, we can make a case study out of it. We will sit down and formulate concrete hypotheses or questions that you investigate as part of this project, and we will compare your approach and results against state-of-the-art practices. I am particularly interested in supervising theses topics related to mutation testing, testing of embedded software, testing safety-critical systems, security testing of mobile apps, anlysis of project repositories to make software development processes more efficient and effective, but I welcome other topic areas.

The method applied is a case study. Case studies follow a systematic approach as outlined in: "Guidelines for conducting and reporting case study research in software engineering" by Per Runeson and Martin Höst –
URL: http://link.springer.com/article/10.1007%2Fs10664-008-9102-8

Important elements of the thesis are literature study, measurement and interviews with experts in the target company.

A mandatory pre-requisite for me to act as supervisor is that there exists a co-supervisor within the case company who is willing to help with the exact scoping of the thesis project and confirms that the topic is in the interest of the case company to an extend that the student can work on the thesis project (at least partly) during work time.

The following topics are research-related and not available for students who work in industry with more than 50% of the nominal workload in industry (more details to come soon):

1. Literature Survey on digital twins for dynamically changing data sets - co-supervised by one of my PhD students.
2. A tool for generating digital data twins using generative AI, e.g., based on large language models like GPT (generative pre-trained transformer) - co-supervised by one of my PhD students.
3. A tool for mapping data collected from an ADS (Automated Driving System) with safety driver on board to test scenarios in simulation-based safety testing of ADS. The data used for the mapping would be collected during a time interval t before an (unplanned) disengagement happened (i.e., the safety driver took over control because he/she thought a dangerous situation emerges that might not be adequately handled by the ADS) - co-supervised by one of my PhD students.

4. Using Regular Expressions (RegEx) to automatically derive applicability constraints for Metamorphic Relations (MR) – for details see below …

Using Regular Expressions (RegEx) to automatically derive applicability constraints for Metamorphic Relations (MR)

Supervisor: Dietmar Pfahl (dietmar dot pfahl at ut dot ee) & Alejandra Duque Torres

According to Wikipedia, "Metamorphic testing (MT) is a property-based software testing technique, which can be an effective approach for addressing the test oracle and, thus, automatic test case generation problem. A Metamorphic Relation (MR) defines for a Software Under Test (SUT) how for a valid input x the output y = SUT(x) must change if x is modified, i.e., what properties must apply to y' = SUT(x') where x' results from applying a specific transformation f on x, i.e., x' = f(x). If for all valid inputs x the resulting outputs y fulfil the given MR, then there is no reason to assume the SUT is faulty (assuming the MR is actually applicable).

The problem with MT is that it is not easy to find a sufficiently large set of applicable MRs to a specific SUT. Much domain knowledge is needed to handcraft good MRs that can be used for generating a strong test suite. Since it is rarely the case that an MR holds universally for all valid inputs x, and to have a richer set of applicable MRs, one might define that an MR must only hold for a subset of all valid inputs (i.e., only for positive numbers or for all numbers but 0).

To overcome these problems, one might try to select suitable MRs from a large set of empirically known MRs. A first and easy to apply filter for automatic selection of MRs from this large set is that only those MRs are chosen that syntactically match the signature (or API) of a function or SUT. Once the syntactically applicable MRs can be automatically applied to the SUT using a fuzzer. For each applied MR, several observations can be made (and recorded): a) For all inputs x, the corresponding outputs y fulfill the properties defined by the MR, or b) For none of the inputs x, the corresponding y fulfill the properties, or c) for some x the corresponding y fulfill and for some they don't fulfill the properties.

The two extreme cases a) and b) can easily exploited for building regression test suites (i.e., after sufficient trust has been built into the correctness of the SUT). However, the MRs that fall under case c) are difficult to reuse, because it is unclear whether the choice of a not yet tried x' must produce an output y' that fulfills or that doesn't fulfill the property defined by the MR. In other words, it is difficult to describe the input space T that corresponds to the partition where y' fulfills the property and what is the input space F that doesn't.

The idea of this thesis topic is to derive RegEx that would define the two partitions of the input space based on the data available from the executed tests using the fuzzer and violation/non-violation of the MR information.

In the context of the Regex Golf challenge similar problems have been tackled (see: https://link.springer.com/article/10.1007/s10710-021-09411-x)

The goal of this thesis project is to devise a method and develop a supporting tool that helps testers to semi-automatically derive regression test suites for a given SUT starting out from a set of MRs and using a fuzzer (see: https://www.fuzzingbook.org). The resulting method and tool must be evaluated.

References:

- de Almeida Farzat, A., de Oliveira Barros, M. Automatic generation of regular expressions for the Regex Golf challenge using a local search algorithm. Genet Program Evolvable Mach 23, 105–131 (2022). https://doi.org/10.1007/s10710-021-09411-x
- T.Y. Chen; S.C. Cheung; S.M. Yiu (1998), "Metamorphic testing: A new approach for generating next test cases", Technical Report HKUST-CS98-01 (PDF), Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong, arXiv:2002.12543.

Bachelor's Thesis

Lab Package Development & Evaluation for the Course 'Software Testing' (LTAT.05.006)

Supervisor: Dietmar Pfahl (dietmar dot pfahl at ut dot ee)

The course Software Testing
(https://courses.cs.ut.ee/2023/SWT2023/spring/Main/HomePage - LTAT.05.006]]) has a series of practice sessions
(https://courses.cs.ut.ee/2023/SWT2023/spring/Main/LabsPracticeSessions)
in which 2nd and 3rd year BSc students learn a specific test technique. We would like to improve existing labs and add new labs.

This topic is intended for students who have already taken this software testing course and who feel that they can contribute to improving it and by the same token complete their Bachelor's project. The scope of the project can be negotiated with the supervisor to fit the size of a Bachelors project.

The tasks to do for this project are as follows:
* Selection of a test-related topic for which a lab package should be developed (see list below)
* Development of the learning scenario (i.e., what shall students learn, what will they do in the lab, what results shall they produce, etc.)
* Development of the materials for the students to use
* Development of example solutions (for the lab supervisors)
* Development of a grading scheme
* Evaluation of the lab package

Topics for which lab packages could be developed (list can be extended based on student suggestions / one bullet point corresponds to one BSc thesis):
* Automatic Test Case Generation (with [[https://www.evosuite.org|EvoSuite]])
* Model-Based Testing (with [[https://graphwalker.github.io|GraphWalker]])
* Fuzzing ([[https://www.fuzzingbook.org|Book]], [[https://en.wikipedia.org/wiki/American_fuzzy_lop_(fuzzer)|AFL]])
* Metamorphic Testing (https://en.wikipedia.org/wiki/Metamorphic_testing)
* Mocking ([[https://site.mockito.org|Mockito]])
* Symbolic Testing (with [[https://github.com/javapathfinder/jpf-core|JPF]])
* Other topics that you find interesting and would like to discuss with me regarding their suitability

---

Using Regular Expressions (RegEx) to automatically derive applicability constraints for Metamorphic Relations (MR)

Supervisor: Dietmar Pfahl (dietmar dot pfahl at ut dot ee) & Alejandra Duque Torres

According to Wikipedia, “Metamorphic testing (MT) is a property-based software testing technique, which can be an effective approach for addressing the test oracle and, thus, automatic test case generation problem. A Metamorphic Relation (MR) defines for a Software Under Test (SUT) how for a valid input x the output y = SUT(x) must change if x is modified, i.e., what properties must apply to y’ = SUT(x’) where x’ results from applying a specific transformation f on x, i.e., x’ = f(x). If for all valid inputs x the resulting outputs y fulfil the given MR, then there is no reason to assume the SUT is faulty (assuming the MR is actually applicable).

The problem with MT is that it is not easy to find a sufficiently large set of applicable MRs to a specific SUT. Much domain knowledge is needed to handcraft good MRs that can be used for generating a strong test suite. Since it is rarely the case that an MR holds universally for all valid inputs x, and to have a richer set of applicable MRs, one might define that an MR must only hold for a subset of all valid inputs (i.e., only for positive numbers or for all numbers but 0).

To overcome these problems, one might try to select suitable MRs from a large set of empirically known MRs. A first and easy to apply filter for automatic selection of MRs from this large set is that only those MRs are chosen that syntactically match the signature (or API) of a function or SUT. Once the syntactically applicable MRs can be automatically applied to the SUT using a fuzzer. For each applied MR, several observations can be made (and recorded): a) For all inputs x, the corresponding outputs y fulfill the properties defined by the MR, or b) For none of the inputs x, the corresponding y fulfill the properties, or c) for some x the corresponding y fulfill and for some they don’t fulfill the properties.

The two extreme cases a) and b) can easily exploited for building regression test suites (i.e., after sufficient trust has been built into the correctness of the SUT). However, the MRs that fall under case c) are difficult to reuse, because it is unclear whether the choice of a not yet tried x’ must produce an output y’ that fulfills or that doesn’t fulfill the property defined by the MR. In other words, it is difficult to describe the input space T that corresponds to the partition where y’ fulfills the property and what is the input space F that doesn’t.

The idea of this thesis topic is to derive RegEx that would define the two partitions of the input space based on the data available from the executed tests using the fuzzer and violation/non-violation of the MR information.

References:

- T.Y. Chen; S.C. Cheung; S.M. Yiu (1998), "Metamorphic testing: A new approach for generating next test cases", Technical Report HKUST-CS98-01 (PDF), Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong, arXiv:2002.12543.
